

Name: _____

CSCI 291 — Spring 2009

Midterm Exam

This exam should have five problems and contain five pages.

Closed book and notes.

Feel free to write helper functions as part of any of your solutions.

Problem 1: [20 points] Our graph coloring algorithm will fail if asked to color a graph in which a node is connected to *itself* by an edge. It would be nice to be able to check for these “self loops”. Write a recursive function called `contains_self_loop` that takes a graph (a list of edges) and returns `True` if an edge in the graph connects a node to itself. You may assume that the `Edge` type is already defined.

```
Main> self_loop [E 5 5]
True
Main> self_loop [E 'x' 'y', E 'y' 'x']
False
Main> self_loop [E 1 2, E 1 3, E 2 2]
True
```

Problem 2: [20 points] Write a recursive function called `index_of`, that returns the zero-based position of an item within a list. For full credit, return an item of type `Maybe`, since there's a possibility that that the item won't be found. If the item appears more than once in the list, return the position of the first occurrence.

```
Main> index_of 5 [1,2,3]
Nothing
Main> index_of 5 [1,5,3,5]
Just 1
Main> index_of "fun" ["this", "exam", "is", "fun"]
Just 3
```

Problem 3: [20 points] Define a function called `impervious` that takes two arguments, a function and a list of values, and returns those items from the list that are “unchanged” by the function — that is, when the function is applied to an item an identical output is produced. For example, in the first test below, the input function is the function that squares its argument. The only values in the list that produce the same output when squared are 0 and 1. For full credit, your solution should not be explicitly recursive. It should instead use higher-order functions like `map`, `filter`, `fold`, or list comprehensions.

```
Main> impervious (\x->x*x) [-2,-1,0,1,2]
[0,1]
Main> impervious (abs) [1,2,3,4]
[1,2,3,4]
Main> impervious (abs) [-2,2,-1,1,0]
[2,1,0]
```

Problem 4: [20 points] Define a function called `sum_all_starting_with` that takes two arguments, an item and a list of lists. It returns the *sum* of all of the lists that begin with the specified item. For example, in the first sample run below, the lists `[1,2]` and `[1]` both begin with 1 (the first argument to the function), so we add up their contents and get 4. (Don't forget that there's a built-in `sum` function that takes lists of numbers and returns their sum.) For full credit, your solution should not be explicitly recursive. It should instead use higher-order functions like `map`, `filter`, `fold`, or list comprehensions.

```
Main> sum_all_starting_with 1 [[1,2], [5,1,2,3], [1], [5,2]]
4
Main> sum_all_starting_with 5 [[1,2], [5,1,2,3], [1], [5,2]]
18
Main> sum_all_starting_with 5 [[1,2], [5,6,7], [5], [4,5]]
23
Main> sum_all_starting_with 9 [[1,2], [5,6,7], [5], [4,5]]
0
```

Problem 5: [20 points]

The questions below apply to this mysterious function. When reading the examples below, don't forget that strings are shorthand notation for lists of characters.

```
mystery things =  
  foldr1 (\x y->if (help x > help y) then x else y) things  
  where  
    help [] = 0  
    help [_] = 1  
    help (n:m:ns)  
      | n == m      = 1 + help (m:ns)  
      | otherwise = 1
```

- 5a) If we could call `help` directly, what would it return when applied to "exam"?
- 5b) If we could call `help` directly, what would it return when applied to "aardvark"?
- 5c) What would `mystery` return if passed ["aardvark", "abbba", "oof"]?
- 5d) Describe, in *English*, what the function does in general.